



CE251 : JAVA PROGRAMMING

Java NIO

Ms. Sachi Joshi
Assistant Professor



**Devang Patel Institute of Advance Technology and
Research**

This Presentation:

- What NIO provides
- A short “tutorial-like” introduction to NIO

What is Java NIO?

- Stands for New IO.
- Started off as a JSR under Sun's JCP (JSR51).

“APIs for scalable I/O, fast buffered binary and character I/O, regular expressions, charset conversion, and an improved filesystem interface.”

Should I Stop Using `java.io`?

- Nope
- `java.nio` is not a replacement for `java.io`
- NIO addresses different needs
- `java.io` is not going away

What Makes Up NIO?

- Buffers
- Channels
- Selectors
- Regular Expressions
- Character Set Coding

How does it do it?

- An API for scalable I/O operations on both files and sockets, in the form of either asynchronous requests or polling;
- An API for fast buffered binary I/O, including the ability to map files into memory when that is supported by the underlying platform;
- An API for fast buffered character I/O, including a simple parsing facility based upon regular expressions and a simple printf-style formatting facility;

JDK 1.4.x

- Provides the package `java.nio`
- This package is a subset of JSR-51. In particular, proposed file-handling functionality is absent

So what does NIO give us?

- A lot – but not everything
- Native code is still required – but you lose portability
- ...but some of the things that could be done before only with native code are now provided in NIO

The java.nio API

- java.nio
 - java.nio.*
 - java.nio.charset and java.nio.charset.spi
 - java.nio.channels and java.nio.channels.spi

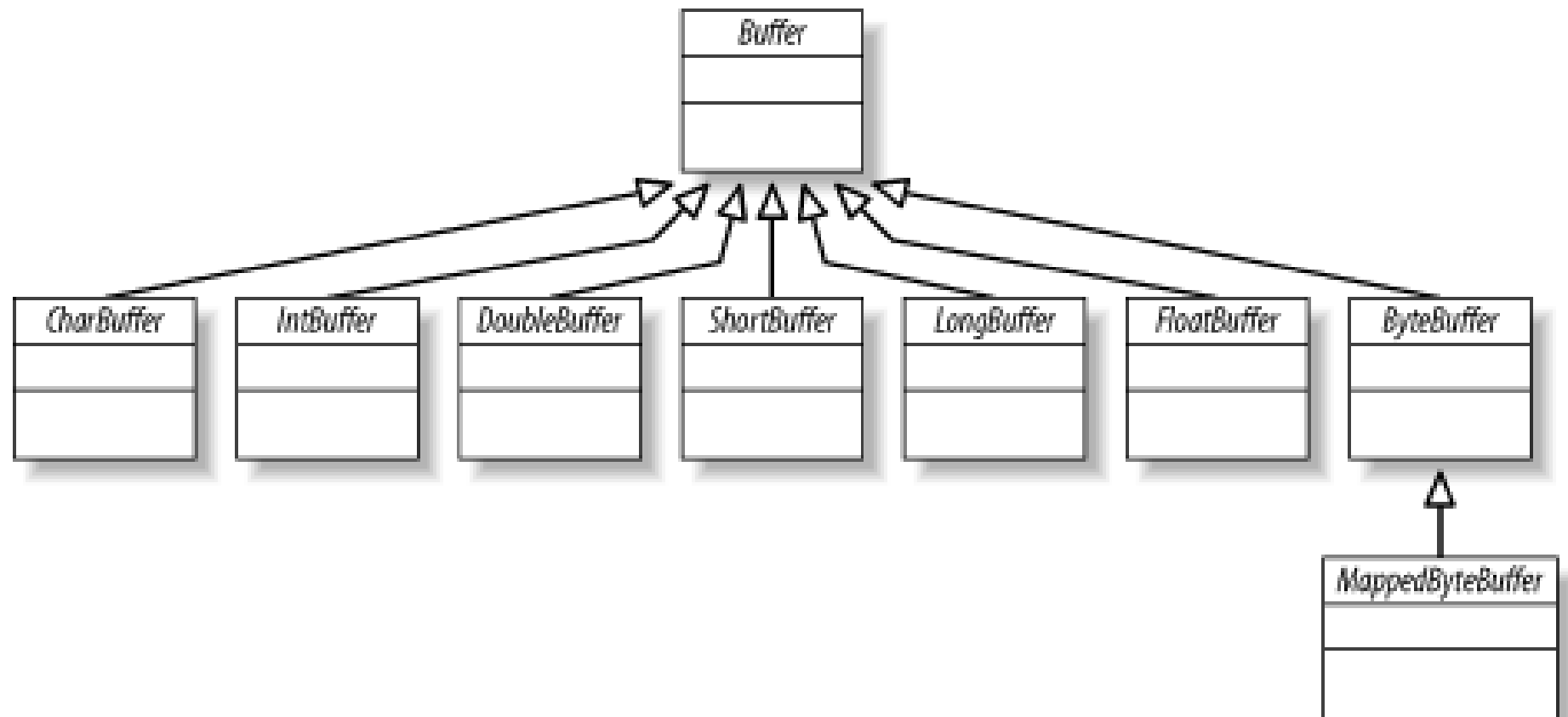
java.nio.*

- A whole set of abstract buffers classes, all extending the abstract class `java.nio.buffer`
- eg: `ByteBuffer`, `CharBuffer`, `DoubleBuffer`, `FloatBuffer`, `IntBuffer`, `LongBuffer`, `ShortBuffer`
- Set of methods: `reset()`, `flip()`, etc.
- $0 \leq \textit{mark} \leq \textit{position} \leq \textit{limit} \leq \textit{capacity}$

NIO Buffers

- Fixed size containers of primitive data types
 - ByteBuffer, CharBuffer, FloatBuffer, etc.
- Byte buffers are special, used for I/O with channels
- Direct and Non-direct ByteBuffers
 - Direct ByteBuffers address raw memory – direct I/O
- Buffers can be views of other buffers or wrap arrays
- Byte order (endian-ness)
 - Affects byte swabbing in views of ByteBuffers

Buffer Classes



Hello NIO Example?

```
import java.nio.ByteBuffer;
import java.nio.channels.WritableByteChannel;
import java.nio.channels.Channels;

public class HelloWorldNio
{
    public static void main (String [] argv)
        throws Exception
    {
        String hello = "Hello World" + System.getProperty
("line.separator");
        ByteBuffer bb = ByteBuffer.wrap (hello.getBytes ("UTF-8"));
        WritableByteChannel wbc = Channels.newChannel (System.out);

        wbc.write (bb);
        wbc.close();
    }
}
```

java.nio.channels

- Selector provider in `.channels.spi` provides the abstract class by which the Java virtual machine maintains a single system-wide default provider instance
- The provider provides instances of `DatagramChannel`, `Pipe`, `Selector`, `ServerSocketChannel`, and `SocketChannel`

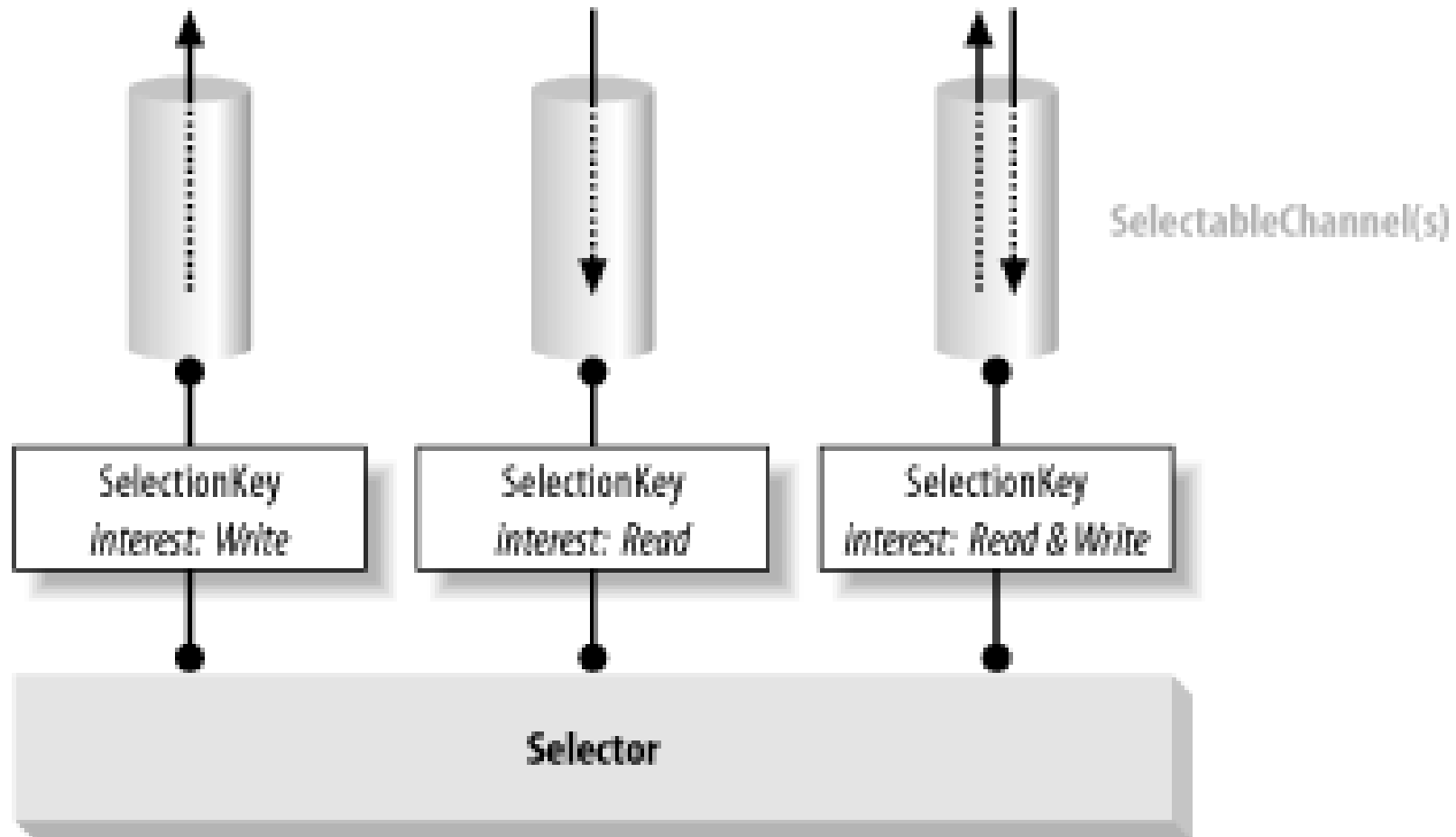
NIO Channels

- New I/O metaphor: Conduit to an I/O service (“nexus”)
- Channels do bulk data transfers to and from buffers
 - `channel.write (buffer) ~= buffer.get (byteArray)`
 - `channel.read (buffer) ~= buffer.put (byteArray)`
- Scatter/gather, channel-to-channel transfers
- Three primary channel implementations
 - ◆ FileChannel: File locks, memory mapping, cross-connect transfers
 - ◆ Sockets: Non-blocking, selectable, async connections, peers
 - ◆ Pipe: loopback channel pair, selectable, generic channels
- Selectable Channel Implementations are pluggable (SPI)

NIO Selectors

- Multiplexing Channels – Readiness Selection
- Selectable Channels are registered with Selectors
 - ◆ SelectionKey encapsulates selector/channel relationship
- A subset of *ready* channels is *selected* from the Selector's set of registered channels (`Selector.select()`)
 - ◆ *Selected Set contains those keys with non-empty Ready Sets*
- Each SelectionKey holds an Interest Set and a Ready Set
 - ◆ *Possible members of Interest Set: accept, read, write, connect*
 - ◆ *Ready set is a subset of interest set –as-of the last `select()` call*
- Readiness Selection means less work – ignore idle channels

Selectors, Keys and Channels

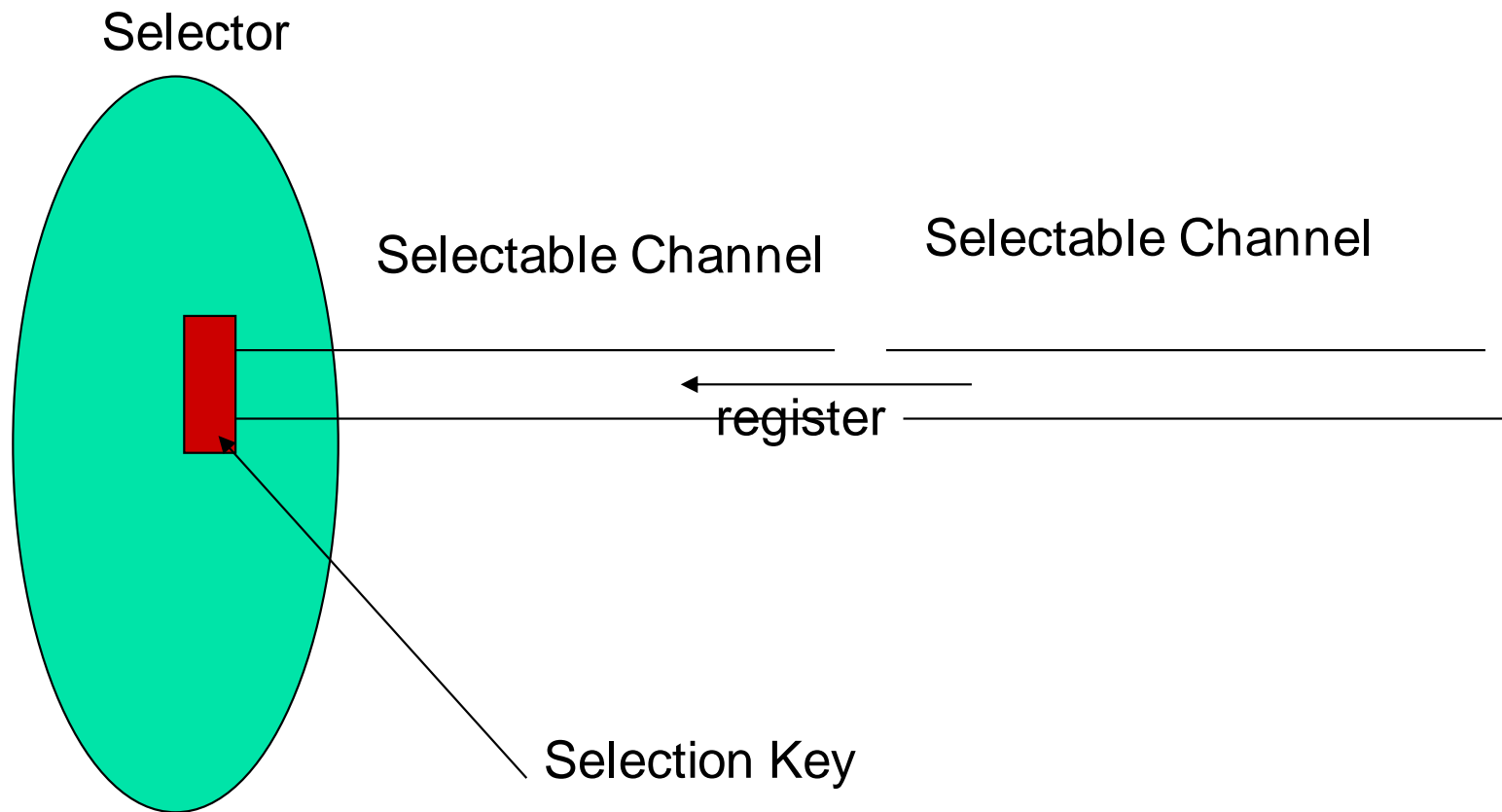


Registering With a Selector

```
ServerSocketChannel serverChannel = ServerSocketChannel.open();  
Selector selector = Selector.open();  
  
serverChannel.socket().bind (new InetSocketAddress (port));  
serverChannel.configureBlocking (false);  
serverChannel.register (selector, SelectionKey.OP_ACCEPT);
```

The Selection Process

- Create a Selector and register channels with it
 - ➔ The `register()` method is on `SelectableChannel`, not `Selector`
- Invoke `select()` on the Selector object
- Retrieve the Selected Set of keys from the Selector
 - ➔ Selected set: Registered keys with non-empty Ready Sets
 - ➔ `keys = selector.selectedKeys()`
- Iterate over the Selected Set
 - ◆ Check each key's Ready Set (set of operations ready to go as-of last `select()`)
 - ◆ Remove the key from the Selected Set (`iterator.remove()`)
 - ➔ Bits in the Ready Sets are never reset while the key is in the Selected Set
 - ➔ The Selector never removes keys from the Selected Set – you must do so
 - Service the channel (`key.channel()`) as appropriate (read, write, etc)



Selectors

- A selector maintains three sets of selection keys
 - The *key set*
 - The *selected-key set*
 - The *cancelled-key*
- Event driven – block for events on selection
- Selection is done by `select()`, `selectNow()` and `select(int timeout)`. Returns the selected key set.

Selection Keys

- A selection key is created each time a channel is registered with a selector
- A selection key contains two *operation sets*
 - Interest Set
 - Ready Set

Conclusion

- Useful and long awaited addition to Java
- Will make message-passing applications more scalable